[MUSIC PLAYING]

PATRICK H. WINSTON: Well, what we're going to do today is climb a pretty big mountain because we're going to go from a neural net with two parameters to discussing the kind of neural nets in which people end up dealing with 60 million parameters. So it's going to be a pretty big jump.

Along the way are a couple things I wanted to underscore from our previous discussion. Last time, I tried to develop some intuition for the kinds of formulas that you use to actually do the calculations in a small neural net about how the weights are going to change. And the main thing I tried to emphasize is that when you have a neural net like this one, everything is sort of divided in each column.

You can't have the performance based on this output affect some weight change back here without going through this finite number of output variables, the y1s. And by the way, there's no y2 and y4-- there's no y2 and y3. Dealing with this is really a notational nightmare, and I spent a lot of time yesterday trying to clean it up a little bit.

But basically, what I'm trying to say has nothing to do with the notation I have used but rather with the fact that there's a limited number of ways in which that can influence this, even though the number of paths through this network can be growing exponential. So those equations underneath are equations that derive from trying to figure out how the output performance depends on some of these weights back here.

And what I've calculated is I've calculated the dependence of the performance on w1 going that way, and I've also calculated the dependence of performance on w1 going that way. So that's one of the equations I've got down there. And another one deals with w3, and it involves going both this way and this way. And all I've done in both cases, in all four cases, is just take the partial derivative of performance with respect to those weights and use the chain rule to expand it.

And when I do that, this is the stuff I get. And that's just a whole bunch of partial derivatives. But if you look at it and let it sing a little bit to you, what you see is that there's a lot of redundancy in the computation. So for example, this guy here, partial of performance with respect to w1, depends on both paths, of course.

But look at the first elements here, these guys right here. And look at the first elements in the expression for calculating the partial derivative of performance with respect to w3, these guys. They're the same.

And not only that, if you look inside these expressions and look at this particular piece here, you see that that is an expression that was needed in order to calculate one of the downstream weights, the changes in one of the downstream weights. But it happens to be the same thing as you see over here. And likewise, this piece is the same thing you see over here.

So each time you move further and further back from the outputs toward the inputs, you're reusing a lot of computation that you've already done. So I'm trying to find a way to sloganize this, and what I've come up with is what's done is done and cannot be-- no, no. That's not quite right, is it? It's what's computed is computed and need not be recomputed. OK? So that's what's going on here. And that's why this is a calculation that's linear in the depths of the neural net, not exponential.

There's another thing I wanted to point out in connection with these neural nets. And that has to do with what happens when we look at a single neuron and note that what we've got is we've got a bunch of weights that you multiply times a bunch of inputs like so. And then those are all summed up in a summing box before they enter some kind of non-linearity, in our case a sigmoid function.

But if I ask you to write down the expression for the value we've got there, what is it? Well, it's just the sum of the w's times the x's. What's that? That's the dot product. Remember a few lectures ago I said that some of us believe that the dot product is a fundamental calculation that takes place in our heads? So this is why we think so. If neural nets are doing anything like this, then there's a dot product between some weights and some input values.

Now, it's a funny kind of dot product because in the models that we've been using, these input variables are all or none, or 0 or 1. But that's OK. I have it on good authority that there are neurons in our head for which the values that are produced are not exactly all or none but rather have a kind of proportionality to them. So you get a real dot product type of operation

out of that.

So that's by way of a couple of asides that I wanted to underscore before we get into the center of today's discussion, which will be to talk about the so-called deep nets. Now, let's see, what's a deep net do? Well, from last time, you know that a deep net does that sort of thing, and it's interesting to look at some of the offerings here.

By the way, how good was this performance in 2012? Well, it turned out that the fraction of the time that the system had the right answer in its top five choices was about 15%. And the fraction of the time that it got exactly the right answer as its top pick was about 37%-- error, 15% error if you count it as an error if it's-- what am I saying?

You got it right if you got it in the top five. An error rate on that calculation, about 15%. If you say you only get it right if it was your top choice, then the error rate was about 37%. So pretty good, especially since some of these things are highly ambiguous even to us.

And what kind of a system did that? Well, it wasn't one that looked exactly like that, although that is the essence of it. The system actually looked like that. There's quite a lot of stuff in there. And what I'm going to talk about is not exactly this system, but I'm going to talk about the stuff of which such systems are made because there's nothing particularly special about this. It just happens to be a particular assembly of components that tend to reappear when anyone does this sort of neural net stuff.

So let me explain that this way. First thing I need to talk about is the concept of-- well, I don't like the term. It's called convolution. I don't like the term because in the second-best course at the Institute, Signals and Systems, you learn about impulse responses and convolution integrals and stuff like that. And this hints at that, but it's not the same thing because there's no memory involved in what's going on as these signals are processed. But they call it convolutional neural nets anyway.

So here you are. You got some kind of image. And even with lots of computing power and GPUs and all that sort of stuff, we're not talking about images with 4 million pixels. We're talking about images that might be 256 on a side. As I say, we're not talking about images that are 1,000 by 1,000 or 4,000 by 4,000 or anything like that. They tend to be kind of compressed into a 256-by-256 image.

And now what we do is we run over this with a neuron that is looking only at a 10-by-10 square

like so, and that produces an output. And next, we went over that again having shifted this neuron a little bit like so. And then the next thing we do is we shift it again, so we get that output right there. So each of those deployments of a neuron produces an output, and that output is associated with a particular place in the image. This is the process that is called convolution as a term of art.

Now, this guy, or this convolution operation, results in a bunch of points over here. And the next thing that we do with those points is we look in local neighborhoods and see what the maximum value is. And then we take that maximum value and construct yet another mapping of the image over here using that maximum value.

Then we slide that over like so, and we produce another value. And then we slide that over one more time with a different color, and now we've got yet another value. So this process is called pooling. And because we're taking the maximum, this particular kind of pooling is called max pooling.

So now let's see what's next. This is taking a particular neuron and running it across the image. We call that a kernel, again sucking some terminology out of Signals and Systems. But now what we're going to do is we're going to say we could use a whole bunch of kernels. So the thing that I produce with one kernel can now be repeated many times like so. In fact, a typical number is 100 times.

So now what we've got is we've got a 256-by-256 image. We've gone over it with a 10-by-10 kernel. We have taken the maximum values that are in the vicinity of each other, and then we repeated that 100 times. So now we can take that, and we can feed all those results into some kind of neural net.

And then we can, through perhaps a fully-connected job on the final layers of this, and then in the ultimate output we get some sort of indication of how likely it is that the thing that's being seen is, say, a mite. So that's roughly how these things work.

So what have we talked about so far? We've talked about pooling, and we've talked about convolution. And now we can talk about some of the good stuff. But before I get into that, this is what we can do now, and you can compare this with what was done in the old days. It was done in the old days before massive amounts of computing became available is a kind of neural net activity that's a little easier to see.

You might, in the old days, only have enough computing power to deal with a small grid of picture elements, or so-called pixels. And then each of these might be a value that is fed as an input into some kind of neuron. And so you might have a column of neurons that are looking at these pixels in your image.

And then there might be a small number of columns that follow from that. And finally, something that says this neuron is looking for things that are a number 1, that is to say, something that looks like a number 1 in the image. So this stuff up here is what you can do when you have a massive amount of computation relative to the kind of thing you used to see in the old days.

So what's different? Well, what's different is instead of a few hundred parameters, we've got a lot more. Instead of 10 digits, we have 1,000 classes. Instead of a few hundred samples, we have maybe 1,000 examples of each class. So that makes a million samples. And we got 60 million parameters to play with.

And the surprising thing is that the net result is we've got a function approximator that astonishes everybody. And no one quite knows why it works, except that when you throw an immense amount of computation into this kind of arrangement, it's possible to get a performance that no one expected would be possible. So that's sort of the bottom line.

But now there are a couple of ideas beyond that that I think are especially interesting, and I want to talk about those. First idea that's especially interesting is the idea of autocoding, and here's how the idea of autocoding works. I'm going to run out of board space, so I think I'll do it right here.

You have some input values. They go into a layer of neurons, the input layer. Then there is a so-called hidden layer that's much smaller. So maybe in the example, there will be 10 neurons here and just a couple here. And then these expand to an output layer like so.

Now we can take the output layer, z1 through zn, and compare it with the desired values, d1 through dn. You following me so far? Now, the trick is to say, well, what are the desired values? Let's let the desired values be the input values. So what we're going to do is we're going to train this net up so that the output's the same as the input.

What's the good of that? Well, we're going to force it down through this [? neck-down ?] piece of network. So if this network is going to succeed in taking all the possibilities here and

cramming them into this smaller inner layer, the so-called hidden layer, such that it can reproduce the input [? at ?] the output, it must be doing some kind of generalization of the kinds of things it sees on its input. And that's a very clever idea, and it's seen in various forms in a large fraction of the papers that appear on deep neural nets.

But now I want to talk about an example so I can show you a demonstration. OK? So we don't have GPUs, and we don't have three days to do this. So I'm going to make up a very simple example that's reminiscent of what goes on here but involves hardly any computation.

What I'm going to imagine is we're trying to recognize animals from how tall they are from the shadows that they cast. So we're going to recognize three animals, a cheetah, a zebra, and a giraffe, and they will each cast a shadow on the blackboard like me. No vampire involved here. And what we're going to do is we're going to use the shadow as an input to a neural net. All right? So let's see how that would work.

So there is our network. And if I just clicked into one of these test samples, that's the height of the shadow that a cheetah casts on a wall. And there are 10 input neurons corresponding to each level of the shadow. They're rammed through three inner layer neurons, and from that it spreads out and becomes the outer layer values. And we're going to compare those outer layer values to the desired values, but the desired values are the same as the input values.

So this column is a column of input values. On the far right, we have our column of desired values. And we haven't trained this neural net yet. All we've got is random values in there. So if we run the test samples through, we get that and that. Yeah, cheetahs are short, zebras are medium height, and giraffes are tall. But our output is just pretty much 0.5 for all of them, for all of those shadow heights, all right, [? with ?] no training so far.

So let's run this thing. We're just using simple [? backdrop, ?] just like on our world's simplest neural net. And it's interesting to see what happens. You see all those values changing? Now, I need to mention that when you see a green connection, that means it's a positive weight, and the density of the green indicates how positive it is. And the red ones are negative weights, and the intensity of the red indicates how red it is.

So here you can see that we still have from our random inputs a variety of red and green values. We haven't really done much training, so everything correctly looks pretty much random. So let's run this thing. And after only 1,000 iterations going through these examples and trying to make the output the same as the input, we reached a point where the error rate

has dropped. In fact, it's dropped so much it's interesting to relook at the test cases.

So here's a test case where we have a cheetah. And now the output value is, in fact, very close to the desired value in all the output neurons. So if we look at another one, once again, there's a correspondence in the right two columns. And if we look at the final one, yeah, there's a correspondence in the right two columns.

Now, you back up from this and say, well, what's going on here? It turns out that you're not training this thing to classify animals. You're training it to understand the nature of the things that it sees in the environment because all it sees is the height of a shadow. It doesn't know anything about the classifications you're going to try to get out of that. All it sees is that there's a kind of consistency in the kind of data that it sees on the input values. Right?

Now, you might say, OK, oh, that's cool, because what must be happening is that that hidden layer, because everything is forced through that narrow pipe, must be doing some kind of generalization. So it ought to be the case that if we click on each of those neurons, we ought to see it specialize to a particular height, because that's the sort of stuff that's presented on the input.

Well, let's go see what, in fact, is the maximum stimulation to be seen on the neurons in that hidden layer. So when I click on these guys, what we're going to see is the input values that maximally stimulate that neuron. And by the way, I have no idea how this is going to turn out because the initialization's all random.

Well, that's good. That one looks like it's generalized the notion of short. Ugh, that doesn't look like medium. And in fact, the maximum stimulation doesn't involve any stimulation from that lower neuron. Here, look at this one. That doesn't look like tall. So we got one that looks like short and two that just look completely random.

So in fact, maybe we better back off the idea that what's going on in that hidden layer is generalization and say that what is going on in there is maybe the encoding of a generalization. It doesn't look like an encoding we can see, but there is a generalization that's-- let me start that over.

We don't see the generalization in the stimulating values. What we have instead is we have some kind of encoded generalization. And because we got this stuff encoded, it's what makes these neural nets so extraordinarily difficult to understand.

We don't understand what they're doing. We don't understand why they can recognize a cheetah. We don't understand why it can recognize a school bus in some cases, but not in others, because we don't really understand what these neurons are responding to.

Well, that's not quite true. There's been a lot of work recently on trying to sort that out, but it's still a lot of mystery in this world. In any event, that's the autocoding idea. It comes in various guises. Sometimes people talk about Boltzmann machines and things of that sort. But it's basically all the same sort of idea.

And so what you can do is layer by layer. Once you've trained the input layer, then you can use that layer to train the next layer, and then that can train the next layer after that. And it's only at the very, very end that you say to yourself, well, now I've accumulated a lot of knowledge about the environment and what can be seen in the environment. Maybe it's time to get around to using some samples of particular classes and train on classes. So that's the story on autocoding.

Now, the next thing to talk about is that final layer. So let's see what the final layer might look like. Let's see, it might look like this. There's a [? summer. ?] There's a minus 1 up here. No. Let's see, there's a minus 1 up-- [INAUDIBLE]. There's a minus 1 up there. There's a multiplier here. And there's a threshold value there.

Now, likewise, there's some other input values here. Let me call this one x, and it gets multiplied by some weight. And then that goes into the [? summer ?] as well. And that, in turn, goes into a sigmoid that looks like so. And finally, you get an output, which we'll z.

So it's clear that if you just write out the value of z as it depends on those inputs using the formula that we worked with last time, then what you see is that z is equal to 1 over 1 plus e to the minus w times x minus T-- plus T, I guess. Right? So that's a sigmoid function that depends on the value of that weight and on the value of that threshold.

So let's look at how those values might change things. So here we have an ordinary sigmoid. And what happens if we shift it with a threshold value? If we change that threshold value, then it's going to shift the place where that sigmoid comes down. So a change in T could cause this thing to shift over that way. And if we change the value of w, that could change how steep this guy is.

So we might think that the performance, since it depends on w and T, should be adjusted in

such a way as to make the classification do the right thing. But what's the right thing? Well, that depends on the samples that we've seen. Suppose, for example, that this is our sigmoid function. And we see some examples of a class, some positive examples of a class, that have values that lie at that point and that point and that point.

And we have some values that correspond to situations where the class is not one of the things that are associated with this neuron. And in that case, what we see is examples that are over in this vicinity here. So the probability that we would see this particular guy in this world is associated with the value on the sigmoid curve.

So you could think of this as the probability of that positive example, and this is the probability of that positive example, and this is the probability of that positive example. What's the probability of this negative example? Well, it's 1 minus the value on that curve. And this one's 1 minus the value on that curve.

So we could go through the calculations. And what we would determine is that to maximize the probability of seeing this data, this particular stuff in a set of experiments, to maximize that probability, we would have to adjust T and w so as to get this curve doing the optimal thing. And there's nothing mysterious about it. It's just more partial derivatives and that sort of thing.

But the bottom line is that the probability of seeing this data is dependent on the shape of this curve, and the shape of this curve is dependent on those parameters. And if we wanted to maximize the probability that we've seen this data, then we have to adjust those parameters accordingly.

Let's have a look at a demonstration. OK. So there's an ordinary sigmoid curve. Here are a couple of positive examples. Here's a negative example. Let's put in some more positive examples over here. And now let's run the good, old gradient ascent algorithm on that. And this is what happens. You've seen how the probability, as we adjust the shape of the curve, the probability of seeing those examples of the class goes up, and the probability of seeing the non-example goes down.

So what if we put some more examples in? If we put a negative example there, not much is going to happen. What would happen if we put a positive example right there? Then we're going to start seeing some dramatic shifts in the shape of the curve. So that's probably a noise point. But we can put some more negative examples in there and see how that adjusts the

curve.

All right. So that's what we're doing. We're viewing this output value as something that's related to the probability of seeing a class. And we're adjusting the parameters on that output layer so as to maximize the probability of the sample data that we've got at hand. Right?

Now, there's one more thing. Because see what we've got here is we've got the basic idea of back propagation, which has layers and layers of additional-- let me be flattering and call them ideas layered on top. So here's the next idea that's layered on top.

So we've got an output value here. And it's a function after all, and it's got a value. And if we have 1,000 classes, we're going to have 1,000 output neurons, and each is going to be producing some kind of value. And we can think of that value as a probability.

But I didn't want to write a probability yet. I just want to say that what we've got for this output neuron is a function of class 1. And then there will be another output neuron, which is a function of class 2. And these values will be presumably higher-- this will be higher if we are, in fact, looking at class 1. And this one down here will be, in fact, higher if we're looking at class m.

So what we would like to do is we'd like to not just pick one of these outputs and say, well, you've got the highest value, so you win. What we want to do instead is we want to associate some kind of probability with each of the classes. Because, after all, we want to do things like find the most probable five.

So what we do is we say, all right, so the actual probability of class 1 is equal to the output of that sigmoid function divided by the sum over all functions. So that takes all of that entire output vector and converts each output value into a probability.

So when we used that sigmoid function, we did it with the view toward thinking about that as a probability. And in fact, we assumed it was a probability when we made this argument. But in the end, there's an output for each of those classes. And so what we get is, in the end, not exactly a probability until we divide by a normalizing factor.

So this, by the way, is called-- not on my list of things, but it soon will be. Since we're not talking about taking the maximum and using that to classify the picture, what we're going to do is we're going to use what's called softmax. So we're going to give a range of classifications, and we're going to associate a probability with each. And that's what you saw in all of those

samples. You saw, yes, this is [? containership, ?] but maybe it's also this, that, or a third, or fourth, and fifth thing.

So that is a pretty good summary of the kinds of things that are involved. But now we've got one more step, because what we can do now is we can take this output layer idea, this softmax idea, and we can put them together with the autocoding idea. So we've trained just a layer up. And now we're going to detach it from the output layer but retain those weights that connect the input to the hidden layer.

And when we do that, what we're going to see is something that looks like this. And now we've got a trained first layer but an untrained output layer. We're going to freeze the input layer and train the output layer using the sigmoid curve and see what happens when we do that.

Oh, by the way, let's run our test samples through. You can see it's not doing anything, and the output is half for each of the categories even though we've got a trained middle layer. So we have to train the outer layer. Let's see how long it takes. Whoa, that was pretty fast. Now there's an extraordinarily good match between the outputs and the desired outputs. So that's the combination of the autocoding idea and the softmax idea.

[? There's ?] just one more idea that's worthy of mention, and that's the idea of dropout. The plague of any neural net is that it gets stuck in some kind of local maximum. So it was discovered that these things train better if, on every iteration, you flip a coin for each neuron.

And if the coin ends up tails, you assume it's just died and has no influence on the output. It's called dropping out those neurons. And in our next iteration, you drop out a different set. So what this seems to do is it seems to prevent this thing from going into a frozen local maximum state. So that's deep nets. They should be called, by the way, wide nets because they tend to be enormously wide but rarely more than 10 columns deep.

Now, let's see, where to go from here? Maybe what we should do is talk about the awesome curiosity in the current state of the art. And that is that all of [? this ?] sophistication with output layers that are probabilities and training using autocoding or Boltzmann machines, it doesn't seem to help much relative to plain, old back propagation. So back propagation with a convolutional net seems to do just about as good as anything.

And while we're on the subject of an ordinary deep net, I'd like to examine a situation here where we have a deep net-- well, it's a classroom deep net. And we'll will put five layers in

there, and its job is still to do the same thing. It's to classify an animal as a cheetah, a zebra, or a giraffe based on the height of the shadow it casts. And as before, if it's green, that means positive. If it's red, that means negative.

And right at the moment, we have no training. So if we run our test samples through, the output is always a 1/2 no matter what the animal is. All right? So what we're going to do is just going to use ordinary back prop on this, same thing as in that sample that's underneath the blackboard. Only now we've got a lot more parameters. We've got five columns, and each one of them has 9 or 10 neurons in it.

So let's let this one run. Now, look at that stuff on the right. It's all turned red. At first I thought this was a bug in my program. But that makes absolute sense. If you don't know what the actual animal is going to be and there are a whole bunch of possibilities, you better just say no for everybody. It's like when a biologist says, we don't know. It's the most probable answer.

Well, but eventually, after about 160,000 iterations, it seems to have got it. Let's run the test samples through. Now it's doing great. Let's do it again just to see if this is a fluke. And all red on the right side, and finally, you start seeing some changes go in the final layers there. And if you look at the error rate down at the bottom, you'll see that it kind of falls off a cliff. So nothing happens for a real long time, and then it falls off a cliff.

Now, what would happen if this neural net were not quite so wide? Good question. But before we get to that question, what I'm going to do is I'm going to do a funny kind of variation on the theme of dropout. What I'm going to do is I'm going to kill off one neuron in each column, and then see if I can retrain the network to do the right thing.

So I'm going to reassign those to some other purpose. So now there's one fewer neuron in the network. If we rerun that, we see that it trains itself up very fast. So we seem to be still close enough to a solution we can do without one of the neurons in each column.

Let's do it again. Now it goes up a little bit, but it quickly falls down to a solution. Try again. Quickly falls down to a solution. Oh, my god, how much of this am I going to do? Each time I knock something out and retrain, it finds its solution very fast.

Whoa, I got it all the way down to two neurons in each column, and it still has a solution. It's interesting, don't you think? But let's repeat the experiment, but this time we're going to do it a little differently. We're going to take our five layers, and before we do any training I'm going to

knock out all but two neurons in each column. Now, I know that with two neurons in each column, I've got a solution. I just showed it. I just showed one.

But let's run it this way. It looks like increasingly bad news. What's happened is that this sucker's got itself into a local maximum. So now you can see why there's been a breakthrough in this neural net learning stuff. And it's because when you widen the net, you turn local maxima into saddle points. So now it's got a way of crawling its way through this vast space without getting stuck on a local maximum, as suggested by this.

All right. So those are some, I think, interesting things to look at by way of these demonstrations. But now I'd like to go back to my slide set and show you some examples that will address the question of whether these things are seeing like we see. So you can try these examples online. There are a variety of websites that allow you to put in your own picture.

And there's a cottage industry of producing papers in journals that fool neural nets. So in this case, a very small number of pixels have been changed. You don't see the difference, but it's enough to take this particular neural net from a high confidence that it's looking at a school bus to thinking that it's not a school bus. Those are some things that it thinks are a school bus.

So it appears to be the case that what is triggering this school bus result is that it's seeing enough local evidence that this is not one of the other 999 classes and enough positive evidence from these local looks to conclude that it's a school bus. So do you see any of those things? I don't. And here you can say, OK, well, look at that baseball one. Yeah, that looks like it's got a little bit of baseball texture in it. So maybe what it's doing is looking at texture.

These are some examples from a recent and very famous paper by Google using essentially the same ideas to put captions on pictures. So this, by the way, is what has stimulated all this enormous concern about artificial intelligence. Because a naive viewer looks at that picture and says, oh, my god, this thing knows what it's like to play, or be young, or move, or what a Frisbee is. And of course, it knows none of that. It just knows how to label this picture.

And to the credit of the people who wrote this paper, they show examples that don't do so well. So yeah, it's a cat, but it's not lying. Oh, it's a little girl, but she's not blowing bubbles. What about this one?

[LAUGHTER]

So we've been doing our own work in my laboratory on some of this. And the way the following set of pictures was produced was this. You take an image, and you separate it into a bunch of slices, each representing a particular frequency band. And then you go into one of those frequency bands and you knock out a rectangle from the picture, and then you reassemble the thing. And if you hadn't knocked that piece out, when you reassemble it, it would look exactly like it did when you started.

So what we're doing is we knock out as much as we can and still retain the neural net's impression that it's the thing that it started out thinking it was. So what do you think this is? It's identified by a neural net as a railroad car because this is the image that it started with. How about this one? That's easy, right? That's a guitar. We weren't able to mutilate that one very much and still retain the guitar-ness of it.

How about this one?

AUDIENCE: A lamp?

PATRICK H. WINSTON: What's that?

AUDIENCE: Lamp.

PATRICK H. WINSTON: What?

AUDIENCE: Lamp.

PATRICK H. WINSTON: A lamp. Any other ideas?

AUDIENCE: [INAUDIBLE].

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: Ken, what do you think it is?

AUDIENCE: A toilet.

PATRICK H. See, he's an expert on this subject.

WINSTON:

[LAUGHTER]

It was identified as a barbell. What's that?

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: A what?

AUDIENCE: Cello.

PATRICK H. WINSTON: Cello. You didn't see the little girl or the instructor. How about this one?

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: What?

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: No.

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: It's a grasshopper. What's this?

AUDIENCE: A wolf.

PATRICK H. WINSTON: Wow, you're good. It's actually not a two-headed wolf.

[LAUGHTER]

It's two wolves that are close together.

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: That's a bird, right?

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: Good for you. It's a rabbit.

[LAUGHTER]

How about that?

[? AUDIENCE: Giraffe. ?]

PATRICK H. WINSTON: Russian wolfhound.

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: If you've been to Venice, you recognize this.

AUDIENCE: [INAUDIBLE].

PATRICK H. WINSTON: So bottom line is that these things are an engineering marvel and do great things, but they don't see like we see.